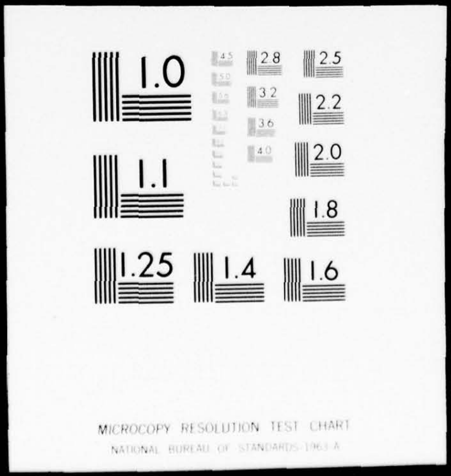
AD-A036 716   MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB   F/G 12/1
A NUMERICAL STUDY OF EIGENVALUE COMPUTATION.(U)
NOV 76   I T YOUNG   F19628-76-C-0002
UNCLASSIFIED   ESD-TR-76-322   NL

1 OF 1

ADA036716



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ESD-TR-76-322

12

Technical Note 1976-32

A Numerical Study of Eigenvalue Computation

I. T. Young

18 November 1976

Prepared for the Department of the Air Force
under Electronic Systems Division Contract F19628-76-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

Approved for public release; distribution unlimited.

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the Department of the Air Force under Contract F19628-76-C-0002.

The views and conclusions contained in this document are those of the contractor and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the United States Government.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Raymond L. Loiselle

Raymond L. Loiselle, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

12

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

A NUMERICAL STUDY OF EIGENVALUE COMPUTATION

I. T. YOUNG, Consultant

Group 69

TECHNICAL NOTE 1976-32

18 NOVEMBER 1976

LEXINGTON MASSACHUSETTS

(See form 1473)

A NUMERICAL STUDY OF EIGENVALUE COMPUTATION

ABSTRACT

A study of different numerical methods for computing the eigenvalues of Hermitian matrices has been carried out. The Givens-Householder procedure was found to be the fastest, most accurate method. Relationships among eigenvalue accuracy, word size, form of arithmetic, hardware implementation, and speed are discussed.

TABLE OF CONTENTS

Table of Contents (Continued)

1.0 INTRODUCTION

In certain communications systems the ability to rapidly and accurately solve the algebraic eigenvalue problem is of extreme importance. Where the location of transmitters is to be determined from a set of N array output readings, one possible signal processing formulation leads to the analysis of an N by N complex, Hermitian* matrix [10]. The elements of this matrix R are complex since sensor readings generate both magnitude and phase information. This matrix R is a correlation matrix formed by taking the expectation E( ) of the matrix $(x_j)(x_j{}^*)^t$ where x_j is a vector of length N corresponding to samples of the N array elements at time j and t represents vector transposition. Thus

$$R = E[(x_j)(x_j{}^*)^t] \quad .$$

Since the signal received at any array element is a linear combination of the transmitted signals from many possible transmitters, the ability to uniquely identify the number of transmitters is limited by the number of array sensors. That is, with N sensors it is only possible to identify at most N supposedly distinct transmitters.

If a given amount of amplification (or power) is available in the form of weights to be assigned to each of the possible array outputs, then at most N-1 weights may be specified independently. Mathematically, if the weights are represented by a complex vector W and the total power available for weighting is constrained, then this is equivalent to the constraint equation

$$W^t W^* = 1 \quad .$$

If we desire that as many of the transmitted signals as possible be nulled, so as to maximize (or minimize) the received power, it is a classical result

*A Hermitian matrix R has elements r_{ij} such that $r_{ij} = r^*_{ji}$; thus the diagonal elements are pure real.

of matrix theory (the Rayleigh quotient) that this can be achieved by choosing as a weight vector the eigenvector of R with the largest (or smallest) eigenvalue. Thus the problem of finding suitable weights for received signal vectors x_j may be recast into the problem of finding the eigenvalues and eigenvectors of the correlation matrix R associated with the vector x_j.

The rapid and accurate computation of these quantities is the subject of this note.

1.1 Algebraic Eigenvalue Problem

The determination of the eigenvalues and eigenvectors for an arbitrary matrix is known as the algebraic eigenvalue problem and its solution in both the theoretical and computational sense has received a considerable amount of attention. The classic discussion of the various techniques available for the solution of this problem is the book by J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press (1965) [1].

The various techniques available read like a flow-chart in terms of the nature of the matrix to be solved. Assuming that the matrix is not degenerate:

1. Is the matrix real or complex?
2. If complex, is it Hermitian?
3. Are only the eigenvalues required or must the eigenvectors be determined as well?
4. How much storage is available?
5. How many bits of accuracy in the results are required?
6. How many operations are required for the computation?

1.2 Hermitian Matrices and Their Solution

The matrices we will be dealing with are complex Hermitian. In general we will need the eigenvectors as well as the eigenvalues for the entire matrix. Possible dynamic strategies when not all the eigenvectors may be required will be discussed in a later section. The reduction of the scope of the problem to this specific type of matrix leads to the conclusion that there are really three major algorithms available for the computation of the eigenvalues and eigenvectors: (1) the Jacobi algorithm, (2) the Hessenberg/QR procedure, and (3) the Givens-Householder algorithm. These algorithms are iterative procedures which converge to the correct answer. They will be discussed in detail in later sections. All of these techniques are based on the following theorem:

> If B is any n × n non-singular matrix, then A and $B^{-1}AB$ have the same eigenvalues. Moreover, if x is an eigenvector of $B^{-1}AB$, then Bx is an eigenvector of A.

This transformation from A to $B^{-1}AB$ is called a similarity transformation. The Jacobi, Hessenberg/QR, and Givens-Householder technique are based on proper choices for the matrix B.

Almost all of the numerical analysis performed on this problem were done on an IBM 370/168 equipped with the basic CPU. For reasons that will be explained later this represents a realistic configuration for a lower bound on the computation for any serial processor. Timing figures for computations performed on this system will be used extensively in the following sections.

2.0 MATRIX ALGORITHMS

The algorithms discussed below have been implemented at the Lincoln Laboratory Computation Center. Complete discussions of them can be found in [5,6,8]. In addition ALGOL versions of the programs are in [7]. Where comparison between the IBM 370/168 and Lincoln Laboratory DEC 11/45 were performed, the identical Givens-Householder FORTRAN routines were run on the same matrices.

2.1 The Jacobi Algorithm

The Jacobi algorithm operates by finding a sequence of similarity transformations such that the transformed matrix converges to a diagonal matrix Λ whose elements on the diagonal represent the eigenvalues (λ_1, λ_2 ..., λ_n). As an example in two dimensions consider the matrix:

$$R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}$$

One possible similarity transformation may be determined by observing that a plane rotation of the coordinates space, (x_1, x_2), to a new space (y_1, y_2) may be performed in such a way that the terms r_{12} and r_{21} are annihilated. This rotation can be represented by

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Expressions for determining the values of $\cos\theta$ and $\sin\theta$ for this annihilation may be found in [2]. The use of this algorithm for Hermitian matrices of size N by N depends on successive rotation of pairs of axes until the sum of the squares of the off-diagonal terms goes to zero (or at least less than some predetermined threshold). The cross-term element r_{pq} that is to be annihilated during any given iteration is called the pivot for the plane rotation and Goldstine, Murray, and Van Neumann [8] showed that if the pivot chosen for each iteration has magnitude greater than the average magnitude for an off-diagonal element then R will converge to Λ.

A computer program for the calculation of the eigenvalues and eigenvectors of a real symmetric matrix using the Jacobi procedure is available in FORTRAN in the IBM Scientific Subroutine Package (EIGEN). Complex Hermitian matrices may be analyzed by noting that R can be written as:

$$R = C + iD \quad \text{where} \quad C, D \in R^{nn}$$

The matrix $\tilde{R} = \begin{bmatrix} C & -D \\ D & C \end{bmatrix}$ which is 2N by 2N may be shown to have eigenvalues $(\lambda_1, \lambda_1, \lambda_2, \lambda_2, \ldots, \lambda_n, \lambda_n)$, where $(\lambda_1, \lambda_2, \ldots \lambda_n)$ are the eigenvalues of R.

Not much more will be said about the Jacobi procedure because it has a major problem in terms of the number of computations required. In general an infinite number of iterations (sweeps) are required to produce a true diagonal matrix. In practice while the residual off-diagonal terms may only have to satisfy some threshold condition (e.g., see above), a term r_{pq} which is annihilated on one iteration will most likely be resurrected on a later iteration. Thus convergence proceeds at a relatively slow rate and the computation time required is large.

2.2 The Hessenberg/QR Algorithm

This procedure represents the concatenation of two procedures for the determination of eigenvalues and eigenvectors. The first procedure reduces a general (not necessarily Hermitian) matrix to a Hessenberg form, that is a matrix H for which $h_{ij} = 0$ if $j < i-1$. Thus for a 4 × 4 matrix we have

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{23} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ 0 & h_{32} & h_{33} & h_{34} \\ 0 & 0 & h_{43} & h_{44} \end{bmatrix}$$

The second procedure uses a series of unitary similarity transformations to reduce the Hessenberg matrix to an upper triangular matrix which displays the eigenvalues on the main diagonal. Let H be the matrix in Hessenberg form to be analyzed, R be an upper triangular matrix and Q a unitary matrix, then if H is non-singular there exists a unique decomposition

$$H = QR$$

The theorem happens to be true for any non-singular matrix H but the number of computations is reduced from N^3 to N^2 if H is a Hessenberg matrix. The process of finding the eigenvalues is based upon application of the iterative equations

$$\begin{aligned} H_i &= Q_i R_i \\ H_{i+1} &= Q_i^{-1} H_i Q_i \\ &= R_i Q_i \quad . \end{aligned}$$

An example of how to determine Q_i and R_i is given in [2, p. 922]. If H_1 (the initial Hessenberg matrix) has eigenvalues satisfying $|\lambda_1| > |\lambda_2| \cdots\cdot > |\lambda_n| > 0$ the sequence $\{H_i\}$ will converge to an upper triangular matrix with the eigenvalues on the main diagonal. A major risk incurred by the use of this routine is that when any pair of eigenvalues are approximately equal, the procedure may not converge.

The use of a reduction-to-Hessenberg algorithm followed by application of the QR procedure is important since it represents a standard technique for the analysis of these complex matrices. At Lincoln Laboratory, the programs may be found in a standard FORTRAN package described in [6] and called by the name EIGEN. Unfortunately, this is the same name used by the IBM Scientific Subroutine Package for the Jacobi algorithm. The number of arguments used in the major subroutine call to EIGEN differs (5 for Hessenberg/QR, 4 for Jacobi), however, making it possible to detect which procedure is being called in any given program.

Finally, it should be noted that although the Hessenberg/QR is considerably faster than the Jacobi procedure and hence is needed to provide baseline computation time measurements, it does not take full advantage of the symmetries inherent in a Hermitian matrix and thus is not the fastest possible algorithm.

2.3 The Givens-Householder Algorithm

The Jacobi algorithm operates to transform a correlation matrix R to a diagonal matrix Λ. The Givens-Householder procedure transforms R to a symmetric tridiagonal matrix, T, whose elements t_{ij} are non-zero for at most $|i-j| \leq 1$. In a sense the T matrix is a very special case of a Hessenberg matrix. Thus after the reduction one possible way to determine the eigenvalues is to apply the QR algorithm. At least two other techniques are available for finding the eigenvalues: the Sturm sequence approach [1,7] and the QL method (a predecessor of the QR method) which is the technique actually used for the work reported here [1,2,7]. The two greatest virtues of the Givens-Householder techniques are (1) its speed -- using the smallest number of computations to get to the point where the eigenvalue/eigenvector routine is actually called, and (2) its excellent numerical stability and accuracy. A detailed explanation of the method by which the symmetric, tridiagonal matrix is produced will not be given here. Instead the reader is referred to [2, p. 901; 3, p. 169].

2.4 Comparison of Methods

The best comparison of the Jacobi to Givens-Householder approach is given in [1, p. 334]. In every aspect (speed, accuracy, storage, solution for a limited number of eigenvalues, etc.) the Givens-Householder approach is superior.

In comparing the Givens-Householder method to the Hessenberg/QR approach, we find that the latter falls between the Jacobi and Givens-Householder method in value. This really is not surprising since in the computation of numerical quantities the fewer the number of computation cycles required the greater the accuracy, thus the fastest algorithm is most often the most accurate. The reason for this is quite clear; every multiply operation and add operation introduces the possibility of noise through either overflow or round-off. Thus the fewer the number of operations the faster and more accurate the result.

In the next section we shall deal with some numerical evaluations of specific matrices and comparisons of accuracies and computations for various algorithms and processors.

3.0 ESTIMATION OF COMPUTATION TIME AND ACCURACY

To compare various algorithms and different machine configurations for computation time and accuracy, a series of experiments were performed.

3.1 Complex vs. Real Arithmetic

Wilkinson has pointed out [7] that it is not uncommon for software packages that implement complex arithmetic to take significantly longer than might be expected. If we assume that a multiply takes approximately five times the length of time required for an add, then a complex multiply should take four to five times the length of an ordinary (real) multiply. It is not unusual, though, to find software packages that do considerably worse than this. To guard against this possibility on the IBM 370/168 a simple test was run to compare the speed of analysis for a 3 × 3 Hermitian matrix $H = A + iB$ against the 6 × 6 real matrix formed by:

$$\begin{bmatrix} A & -B \\ B & A \end{bmatrix}$$

where

$$A = \begin{bmatrix} 5 & 1 & -2 \\ 1 & 6 & -3 \\ -2 & -3 & 8 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 2 & -5 \\ -2 & 0 & -6 \\ 5 & 6 & 0 \end{bmatrix}$$

Using the fastest possible algorithm (i.e., the Givens-Householder) the complex analysis took 276 clock times versus the real analysis which took 1138 clock times (each clock interval equals 13 microseconds). The complex matrix packages may therefore be used; the analysis of N × N complex matrices is significantly faster than the analysis of 2N × 2N real matrices.

3.2 *Hessenberg/QR vs. Givens-Householder Computation Times*

The two procedures were run on identical matrices with the size of the matrix N being varied over the range $2 \leq N \leq 20$. The number of clock times from the start to the finish of each matrix analysis was recorded. A polynomial regression program was used to fit the computation time as a function of matrix size, T(N). A cubic polynomial was found to provide the best fit. This was to be expected since analysis of the computer programs predicts a computation time proportional to N^3. For the data collected the comparative graphs using the cubic fit are given in Fig. 1. The equations for the two curves are:

$$\text{Householder -- } R(N) = 2.64N^3 - 23.54N^2 + 253.26N - 375.03$$

$$\text{Hessenberg -- } T(N) = 29.48N^3 - 358.43N^2 + 1905.63N + 2592.54$$

where computed values of T(N) are in clock intervals of 13 microseconds. While this is not meant to provide exact prediction of computation time for arbitrary Hermitian matrices, it does show an approximate 11:1 improvement in speed for large N. Further the Householder procedure clearly dominates for $N \geq 3$. The coefficient of fit of the two cubic polynomials to the data was given by the correlation coefficient squared which in each case was:

$$\text{Householder -- } \rho^2 = .997$$

$$\text{Hessenberg -- } \rho^2 = .999$$

Only the values of computation time for $2 \leq N \leq 12$ were used in the actual regression analysis. The actual values for $N \geq 14$ may be used to check the prediction accuracy of the cubic polynomials. The cubic equation for the Householder based on $2 \leq N \leq 12$ seems to underpredict the computation time by at most 50% for $14 \leq N \leq 20$. The Hessenberg equation overpredicts the time by as much as 100%. In any case the Householder procedure still clearly dominates the Hessenberg.

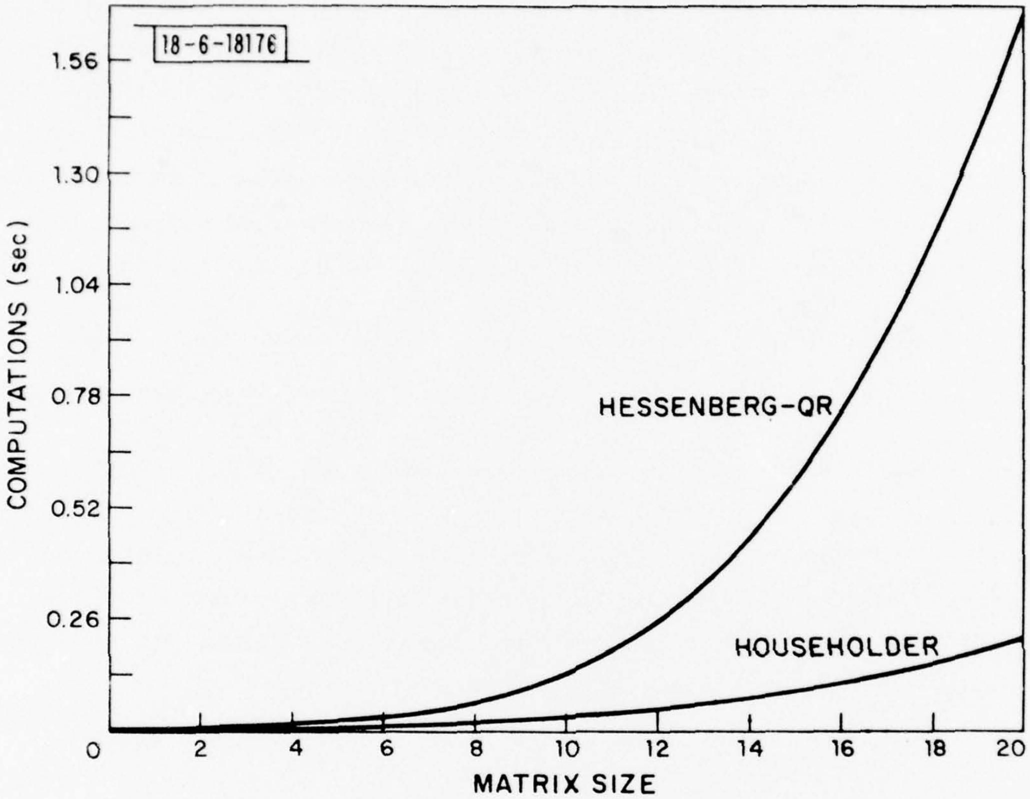


Fig. 1. Comparison of time to compute eigenvectors and eigenvalues of complex Hermitian matrices using two techniques -- Hessenberg/QR and Householder. The latter technique offers approximately an 11:1 improvement in speed for large matrix sizes.

	N = 14	N = 16	N = 18	N = 20	
Householder	9012	14219	19641	25146	actual
	5812	8480	11976	16426	predicted
Hessenberg	24977	38874	52804	66753	actual
	34730	56894	87510	127996	predicted

3.3 Computational Accuracy of the Two Procedures

According to Wilkinson [1] each procedure is numerically quite stable. To compare accuracies the following matrix was analyzed:

$$H = \begin{bmatrix} 5 + 0i & 1 + 2i & -2 - 5i \\ 1 - 2i & 6 + 0i & -3 - 6i \\ -2 + 5i & -3 + 6i & 8 + 0i \end{bmatrix}$$

The three eigenvalues were identical through at least the first 14 decimal places, and this exceeded the published accuracy of these eigenvalues in Wilkinson [7]. Similar, high accuracy results were obtained in other matrices.

3.4 Analysis of Computational Accuracy

Wilkinson has done extensive research into finding closed form expressions for the accuracy of eigenvalues computed by various procedures. In this section we shall deal with the accuracy of the Jacobi procedure and the accuracy of the Householder procedure. We know from our previous discussion that the results for the Hessenberg will fall somewhere between these two.

3.4.1 The Jacobi Procedure

3.4.1.1 Floating-Point Calculations

It has been shown that the RMS error in the estimate of the eigenvalues is given by [1, p. 279]:

$$\varepsilon_{RMS} = \left[\frac{\sum_{i=1}^{N} (\mu_i - \lambda_i)^2}{\sum_{i=1}^{N} \lambda_i^2}\right]^{1/2} \leq 108 \cdot 2^{-t} N^{3/2} \cdot (1 + 9 \cdot 2^{-t})^{6(4N-2)}$$

where:

N = matrix size

t = bits of mantissa

λ_i = exact eigenvalues

μ_i = computed eigenvalues

and 6 sweeps of the Jacobi procedure are assumed. For reasonable size t (say $t \geq 10$) we have:

$$(1 + 9.2^{-t}) \approx 1$$

so that:

$$\varepsilon_{RMS} \leq 108 \cdot 2^{-t} \cdot N^{3/2}$$

But this is the most pessimistic estimate of the error and if we expect a reasonable distribution of rounding errors then a more realistic bound is:

$$\boxed{\varepsilon_{RMS} \leq 11 \cdot 2^{-t} \cdot N^{3/4}}$$

3.4.1.2 Fixed-Point Calculation

For fixed-point computation of the eigenvalue a similar analysis leads to:

$$\boxed{\varepsilon_{RMS} \leq 6K_1 2^{-t} N^{5/2}}$$

where:

K_1 is a constant such that $1 \leq K \leq 10$

t = number of bits

N = matrix size

3.4.2 The Householder Procedure

The error generated in this method may be broken into two pieces corresponding to each of the two major steps in the analysis. The first is the error associated with the computation of the tridiagonal matrix from the original matrix. The second is the error associated with the eigenanalysis of the tridiagonal matrix.

3.4.2.1 Error for Tridiagonal Reduction

3.4.2.1.1 Floating-Point Calculation

Floating-point calculations may be done either with or without accumulation of partial sums in double precision. That is, if we are computing a dot product of two real vectors $(\vec{x} \cdot \vec{y})$, then the computation of:

$$\sum_{i=1}^{N} x_i y_i$$

may be done by rounding-off after each multiply and add operation (without accumulation) or by waiting until the N multiplies and adds are completed and then rounding-off (with accumulation). If accumulation is used the only double precision number that must be saved is the partial sum (of products). The difference in the RMS error between these two techniques is at worst a factor of N. The expressions are given by:

Floating-Pt. with Accumulation

$$\boxed{\varepsilon_{RMS} \leq 40N2^{-t}}$$

Floating-Pt. without Accumulation

$$\boxed{\varepsilon_{RMS} \leq 32N^{2}2^{-t}}$$

The constants 40 and 32 are upper bounds and actually depend on details of how the floating-point math is done. For example, small numbers should be added together first.

3.4.2.1.2 Fixed-Point Calculation

In fixed-point computations accumulation with double precision of partial sums may likewise be used. Unfortunately no simple expression is available for the RMS error. Instead, however, a simple expression is available for the maximum error. The results are

Fixed-Pt. with Accumulation

$$\boxed{\varepsilon_{max} = \max_{i} |\lambda_i - \mu_i| < K_2 N^{2} 2^{-t}}$$

Fixed-Pt. without Accumulation

$$\boxed{\varepsilon_{max} < K_3 \, N^{5/2} 2^{-t}}$$

K_2 and K_3 are constants that once again depend on details of how the mathematics is done and the assumption about round-off error (i.e., worst-case, average, etc.). In either case $1 \leq K_2, K_3 \leq 10$. We will, in fact, use $K_2 = K_3 = 2^4$ as a conservative upper bound.

3.4.2.2 Eigenvalue Error for Tridiagonal

As mentioned earlier, a number of techniques exist for determining the eigenvalues of a tridiagonal matrix. For each of these methods, however, the error is less than the error associated with determining the tridiagonal matrix. In the Sturm sequence approach, for example, the error after p iterations is given by:

$$|\mu_{p+1} - \lambda| < (15.06)2^{-t} + 3 \cdot 2^{-p}$$

This is less than the error for the tridiagonal matrix by a factor of N. Thus we may conclude that the error associated with the Householder method is dominated by the error involved in the reduction to a tridiagonal matrix.

3.4.3 Summary and Comparison

For the Jacobi procedure we have:

		References
Floating Pt.	$\varepsilon_{RMS} \leq 11 \cdot 2^{-t} N^{3/4}$	[1, p. 280]
Fixed Pt.	$\varepsilon_{RMS} \leq 6K_1 2^{-t} N^{5/2}$	[1, p. 281]

For the Householder:

Floating Pt. with accumulation	$\varepsilon_{RMS} \leq 40 \cdot 2^{-t} N$	[1, p. 297]
without accumulation	$\varepsilon_{RMS} \leq 32 \cdot 2^{-t} N^2$	[1, p. 298]

Fixed Pt. with accumulation	$\varepsilon_{max} < K_2 2^{-t} N^2$	[1, p. 298]
without accumulation	$\varepsilon_{max} < K_3 2^{-t} N^{5/2}$	[1, p. 299]

In comparing Jacobi to Householder we see they are virtually the same in terms of the bounds. Thus we will focus our attention solely on the issue of the error in the Householder approach.

While the application referred to in Section 1 needs the solution to the eigenvector problem to determine the appropriate weight vectors, we will use the eigenvalue problem to gain insight into what type of processor power is required to achieve an arbitrary accuracy.

If we assume that we would like to be able to specify the eigenvalue to 12 bits, i.e., one part in 4096, then, how many bits t must the processor have if the matrix size is N? All of our error bounds are of the form:

$$\varepsilon < A2^{-t}N^{\nu}$$

If we require that

$$A2^{-t}N^{\nu} < 2^{-13}$$

then we certainly satisfy the conditions of our problem. Solving for t yields:

$$\boxed{t > 13 + \nu \log_2 N + \log_2 A}$$

Plotting this expression as a function of N with the various strategies as parameters we have Fig. 2. We see that in the worst case (fixed pt., without accumulation) we need less than 30 bits to achieve the required accuracy when N=20. For the best case (floating pt., with accumulation) we need more than 20 bits when N=20. This immediately suggests that the optimum processor (16 bit) in terms of cost and speed uses double precision integer (i.e., fixed pt.) arithmetic. In terms of error analysis this is just as effective a strategy as

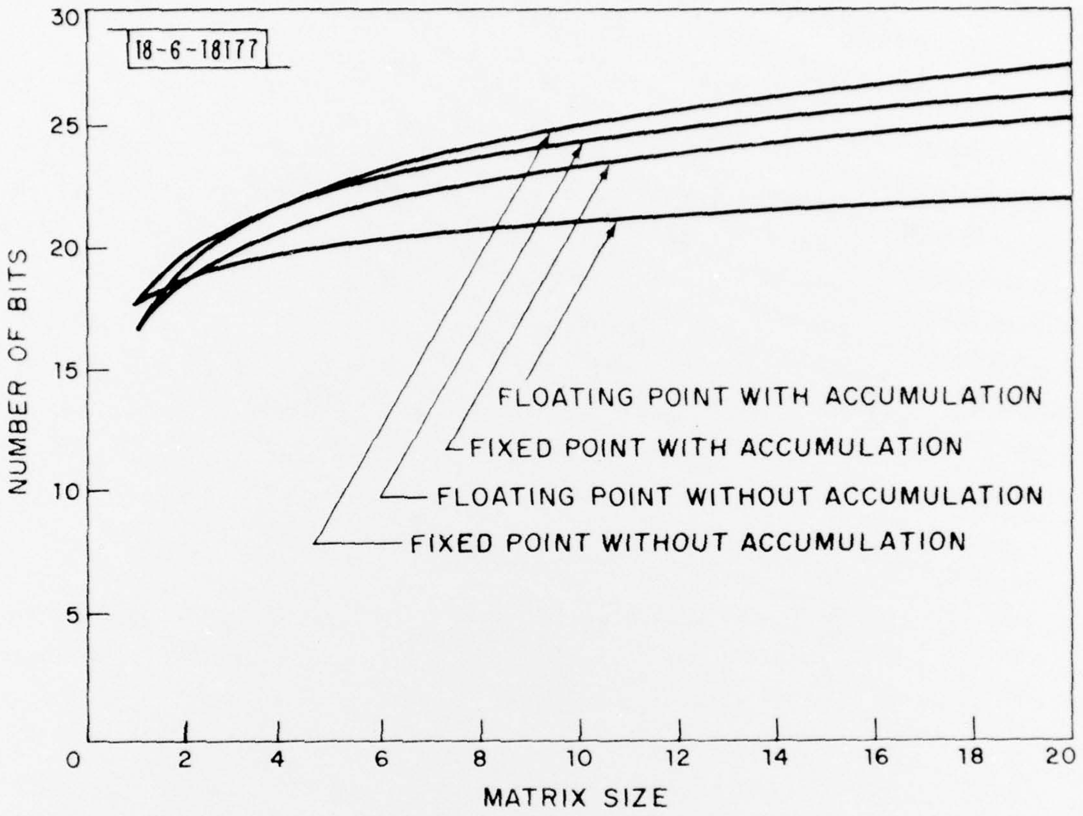


Fig. 2. Comparison of the number of bits theoretically required to achieve 12-bit accuracy in the eigenvalue computation as a function of matrix size and the computational mathematics used.

using floating point with accumulation since it really is not possible to buy a 20 bit processor <u>and</u> floating point computations take significant time even in hardware.*

4.0 REAL-TIME EIGENANALYSIS

It is of interest to determine what is the largest matrix that could be continually analyzed in a given amount of time. For example, if 100 milliseconds are available, what is the largest N such that the complete eigenanalysis could be performed? In that way the weight (eigenvector) could be updated 10 times per second. To answer this we first look at the results from Section 3.2 for the Householder analysis. Replotted as in Fig. 3 we have that in 100 ms the largest value is N=15. This value is clearly dependent on the hardware configuration of the processor and the efficiency of the language translator.

4.1 The IBM 370/168 as a Serial Processor

The computations described above were made on the Lincoln Laboratory IBM 370/168. This machine has a basic machine cycle time of 80 nanoseconds operating out of its 16 general registers. Access to high-speed buffer storage takes from 160 ns (if overlap is used) to 240 ns. Add time is 1 machine cycle. The data path for all these operations is 8 bytes (64 bits) wide. The fixed-point multiply takes 780 ns; the floating-point multiply, 1870 ns. Since there is no special purpose hardware for vector or matrix operations (that is, we cannot do a parallel $\vec{x} \cdot \vec{y}$), we consider for the purpose of this report that the computer is strictly serial processing. No parallelism is available to cut down computation time. In that case the times sited above represent just about the best one can do with current solid-state hardware including special purpose processors. (We are not including design involving the recently announced sub-nonosecond ECL logic.) Thus the 370/168 represents a realistic lower bound for the class of all general purpose and special purpose serial processors.

*It might be useful, however, to explore one or two of the 24-bit machines with specially constructed high-speed floating point hardware with double-precision for accumulation.

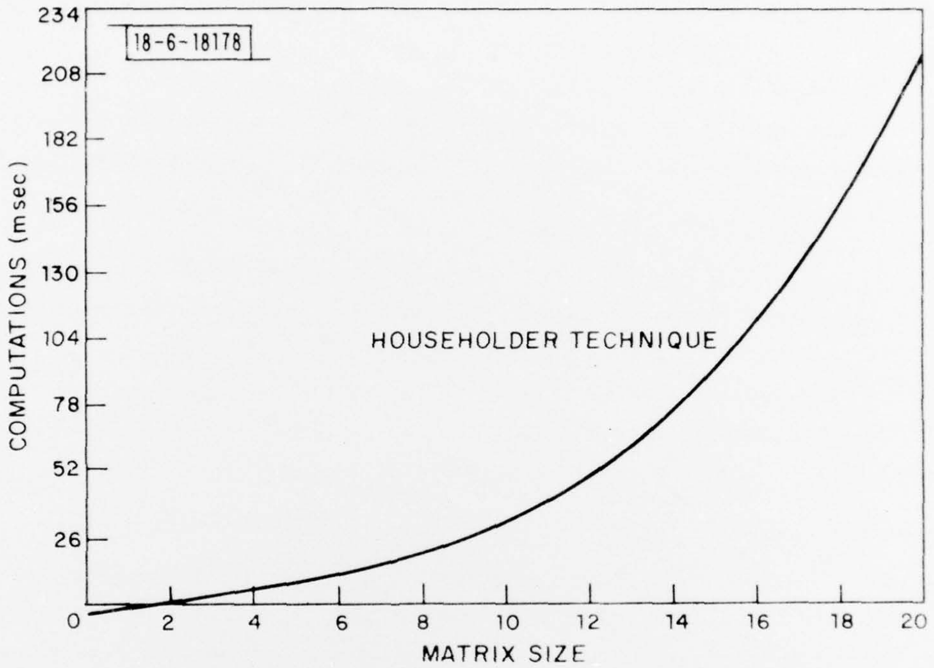


Fig. 3. Time to compute eigenvectors and eigenvalues for a complex Hermitian matrix as a function of matrix size using the Householder algorithm.

4.2 Language Translation

To take advantage of the 370/168 hardware speed the programs executed on it must make efficient use of the general purpose registers, index registers, high-speed buffer storage, overlapping, and hardware multipliers. All of the programs were written in FORTRAN IV and compiled on H EXTENDED IBM OS COMPILER [4]. This compiler has optimization features to improve run-time performance. The programs themselves were carefully coded to maximize performance.

4.3 Summary for IBM Analysis

Since we have here the coupling between an efficient, optimized program and compiler and an extremely high-speed processor we may treat the results in Section 4.0 as a realistic upperbound to the matrix size N that can be processed in a given time T. To go to the other end of the spectrum we run the same programs on a PDP 11/45 to estimate the "slowest" that one would have to settle for in terms of modern minicomputer performance.

4.4 PDP 11/45 System

The DEC PDP 11/45 system used is in Lincoln Laboratory's Group 24. It consists of 64K words of 900 ns core memory and a number of I/O peripherals. It does not include a hardware floating-point processor nor an optimized FORTRAN compiler. Memory interleaving permits an effective cycle time of 450 ns. Running the same matrices and the same programs on the 11/45 as were run on the 370/168 showed a decrease in speed by about a factor of 150. Thus only a 2 × 2 matrix (N=2), could be analyzed in 100 ms. If we compare the largest matrix (N=20) and ask for the total running time on each of the two systems the results would be:

IBM 370/168	(N=20) — $T \cong .25s$
DEC PDP 11/45	(N=20) — $T \cong 40s$

Of course, computation on the 11/45 could be substantially improved by the following:

1. Use of semiconductor memory, and
2. Floating-point hardware and optimized FORTRAN, or
3. Double-precision integer hardware and optimized machine language code.

It is the author's experience that use of the third approach often leads to improvement on the order of 100 in speed thus bringing it close to the IBM system in performance. Ultimately this is at the sacrifice of accuracy, but as discussed in an earlier section the issue of accuracy is not critical here given 32 bit arithmetic.

5.0 ALTERNATIVES FOR IMPROVEMENT OF SYSTEM PERFORMANCE

Two basic approaches are available to improve system performance. The first involves a greater use of parallel processing to perform the eigenanalysis. The second involves a modification of the problem statement to a dynamic strategy so that not all eigenvectors are continually being determined.

5.1 Parallel Processing

The Householder approach naturally suggests the use of pipeline processing techniques to achieve an increase in speed. Two processors can be used; one to perform the reduction to tridiagonal form and the other to perform the eventual eigenanalysis. At best, however, this leads to a decrease in time by a factor of 2 if both processes take the same time. Given the low-cost of modern processors such as the 11/45 and the use of shared memories this is certainly a reasonable first improvement.

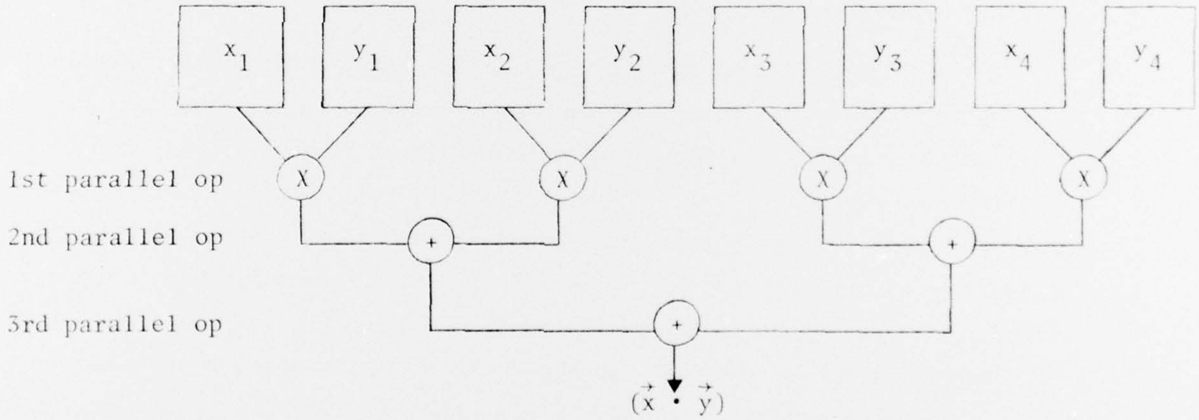
Reviewing the details of the algorithm provides another approach to decreasing the computation time. At a number of places in this memorandum we have used the vector dot product (VDP) as an example of a typical calculation. In fact, the VDP is used extensively in the Householder analysis and this leads us to consider if we might improve its performance on our target machine. The time required (t_{VDP}) is obviously on the order of N times the time required for one multiply and add operation (MAD):

$$t_{VDP} \sim N(MAD)$$

To improve this number we have two choices:

1. Build a special purpose dot-product calculator that is given the starting address and length of the two vectors and uses direct memory access (DMA) to acquire and compute the required result. Instead of N multiplies and adds we have N memory fetches (assuming a super high-speed multiplier in the special purpose calculator). Since a MAD takes about 7 to 9 memory cycles this implies a time reduction of that amount.

2. Build a full-parallel vector dot-product calculation that has special simultaneous access to all the elements of each vector. Assuming a full set of multipliers (N) the number of cycles would be proportional to $\log_2 N$ adds (this is based on an initial model that looks like the following for $(\vec{x} \cdot \vec{y})$ when N=4).



Of course, this is the costliest solution but it provides for reduction of computations from the N^3 noted in an earlier section to about N^2. For large N this can mean a significant reduction in computation time.

5.2 Dynamic Strategies

In many instances where the eigenvector weights that we are determining are not all continually changing, it is inappropriate to continue solving for all the eigenvectors. Alternatively we might only wish to solve for the eigenvector associated with the biggest (or smallest) set of eigenvectors.

Where these two possibilities exist the use of the Sturm sequence approach [1-3,7] provides the solution to just the eigenvalues at a further increase in speed. The desired eigenvectors may then be determined using several different techniques (inverse iteration, power method with deflation, [3]). If the number of eigenvectors that need to be found is small compared to N, then a further savings in computation time will exist. If, however, we do not know, a priori, how many eigenvectors we will need until we have looked at the eigenvalues, then we can only talk about the expected computation time. The expected computation time requires a probabilistic model for the number of eigenvalues that will need to be solved at any time. The determining of an appropriate model could form the basis for further work on this problem. Complicated strategies, like following the trend in the number of eigenvectors that need determination, might also provide incremental savings in computation time.

6.0 SUMMARY AND CONCLUSIONS

1. The Givens-Householder procedure for the analysis of Hermitian matrices provided the fastest, most accurate method for determining the eigenvectors and eigenvalues.

2. Given a requirement of accuracy to 12 bits, 32 bits represent a convenient size for mathematical operation; that is, double precision integer on a 16 bit minicomputer.

3. If the entire operation is coded in machine language and uses hardware multiply/divide then one can expect to be able to process 10 × 10 (N=10) matrices in 100 ms intervals. The size of other matrices that may be

processed given other time intervals may be determined from Fig. 3 and the results of Sections 4.4 and 5.1.

4. If higher speed is required then more parallel processing must be used. In addition dynamic strategies may be employed when only a limited number of eigenvectors need be determined.

REFERENCES

1. J. H. Wilkinson, The Algebraic Eigenvalue Problem (Clarendon Press, Oxford, 1965).

2. D. M. Young and R. T. Gregory, A Survey of Numerical Mathematics (Addison-Wesley, Reading, Massachusetts, 1973).

3. A. M. Cohen, Numerical Analysis (Wiley and Sons, New York, 1973).

4. "IBM OS Fortran IV (H Extended)," Compiler Programmer's Guide, Third Edition (1974).

5. IBM Scientific Subroutine Package, Fifth Edition (1970).

6. Additions to CMS User's Guide, Technical Memorandum S-0112-2, Lincoln Laboratory, M.I.T. (1 January 1975).

7. J. H. Wilkinson and C. Reinsch, Linear Algebra (Springer-Verlag, Berlin, 1971).

8. International Mathematical and *Statistical Library, Library 1, Edition* 4 (FORTRAN IV) S/370-360 (1975).

9. H. H. Goldstine, F. J. Murray, and J. von Neumann, "The Jacobi Method for Real Symmetric Matrices," J. Assoc. Comp. Mach. 6, 59 (1959).

10. Navy Communications Quarterly Technical Summary, Lincoln Laboratory, M.I.T. (15 April 1974), DDC AD-919549-L.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

19 **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
18 ESD-TR-76-322

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (*and Subtitle*)
6 A Numerical Study of Eigenvalue Computation.

5. TYPE OF REPORT & PERIOD COVERED
9 Technical Note,

6. PERFORMING ORG. REPORT NUMBER
Technical Note 1976-32

7. AUTHOR(*s*)
10 Ian T. Young

8. CONTRACT OR GRANT NUMBER(*s*)
15 F19628-76-C-0002

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Lincoln Laboratory, M. I. T.
P. O. Box 73
Lexington, MA 02173

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Program Element Nos. 65705F
and 63431F
Project Nos. 649L and 2029, 16 649L

11. CONTROLLING OFFICE NAME AND ADDRESS
Air Force Systems Command, USAF
Andrews AFB
Washington, DC 20331

12. REPORT ~~DATE~~
11 18 Nov[redacted]976

13. ~~NUMBER OF~~ PAGES
12 32 p.

14. MONITORING AGENCY NAME & ADDRESS (*if different from Controlling Office*)
Electronic Systems Division
Hanscom AFB
Bedford, MA 01731

15. SECURITY CLASS. (*of this report*)
Unclassified

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

eigenvalues	computational accuracy	Jacobi procedure
Hermitian matrices	Givens-Householder procedure	eigenanalysis
matrix algorithms		

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

A study of different numerical methods for computing the eigenvalues of Hermitian matrices has been carried out. The Givens-Householder procedure was found to be the fastest, most accurate method. Relationships among eigenvalue accuracy, word size, form of arithmetic, hardware implementation, and speed are discussed.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

207 650